Chapter 17
Boosting Metaheuristic Search Using Reinforcement Learning

Tony Wauters, Katja Verbeeck, Patrick De Causmaecker, and Greet Vanden Berghe

Abstract. Many techniques that boost the speed or quality of metaheuristic search have been reported within literature. The present contribution investigates the rather rare combination of reinforcement learning and metaheuristics. Reinforcement learning techniques describe how an autonomous agent can learn from experience. Previous work has shown that a network of simple reinforcement learning devices based on learning automata can generate good heuristics for (multi) project scheduling problems. However, using reinforcement learning to generate heuristics is just one method of how reinforcement learning can strengthen metaheuristic search. Both existing and new methodologies to boost metaheuristics using reinforcement learning are presented together with experiments on actual benchmarks.

17.1 Introduction

Researchers developing search methods to solve combinatorial optimization problems are faced with a number of challenges. An important challenge is to avoid convergence to a local optimum. A second challenge is to create a method applicable to different problems of various sizes and properties, while still being able to produce good quality solutions in a short amount of time. Metaheuristics [17, 30] and the more recently introduced hyper-heuristics [6] try to address these issues. Hybrid systems and their various perspectives cope with these challenges even better. One possible hybridization, which is the main topic of this contribution, involves the inclusion of a Reinforcement Learning (RL) [19, 29] component to these meta- and hyper-heuristic methods. This idea fits in the area of intelligent optimization [2],

Tony Wauters
CODeS, KAHO Sint-Lieven, Gebroeders Desmetstraat 1, 9000 Gent, Belgium
e-mail: tony.wauters@kahosl.be

E.-G. Talbi (Ed.): Hybrid Metaheuristics, SCI 434, pp. 433–452.
springerlink.com

where some intelligent (learning) component aids the optimization method in order to obtain a better informed search. During the search process a learning component can adjust parameters or support the optimization method in making decisions. Intelligent optimization can be defined as a combination of techniques from Operations Research and Artificial Intelligence.

In what follows, several arguments for combining RL and metaheuristics are enlisted. Reinforcement learning causes the algorithm to be adaptive, and as such it minimizes the weaknesses of strongly parameterized methods. As long as the algorithm is granted enough time facilitating the learning of valuable information. Reinforcement learning offers interesting advantages. It does not require a complete model of the underlying problem. RL methods learn the model by gathering experience, often referred to as trial-and-error. Many model free RL methods exist. This is noteworthy since ordinarily no model is available for most combinatorial optimization problems. Some reinforcement learning methods can handle incomplete information, although this is obviously a much harder learning task. Reinforcement learning permits applying independent learning agents, and thus, it is applicable for fully decentralized problems. Furthermore, it is computationally cheap, i.e. often it uses only a single update formula at each step. Additionally, if only general problem features and no instance specific features are used, then the learned information can possibly be transfered to other instances of the same problem. Recently some type of RL algorithms that are well suited for this task have been introduced, these are called transfer learning [32, 31]. Lastly, one can build on theoretical properties showing that many RL methods converge to optimal state-action pairs under certain conditions (e.g. the policy for choosing the next action is ergodic) [29]. In this chapter we will show that these interesting theoretical properties show good results in practice.

The remainder of the chapter describes the combination of reinforcement learning and search in more detail. Section 17.2 gives a short introduction to reinforcement learning and some common RL algorithms. Section 17.3 discusses different opportunities for combining these two methods. Section 17.4 gives an extensive literature overview of the combination of learning and search. An example of a succesfull application of RL is presented in Sect. 17.5. A conclusion and some future prospects are described in Sect. 17.6.

17.2 Reinforcement Learning

Reinforcement Learning (RL) [19, 29] is the computational task of learning what action to take in a given situation (state) to achieve one or more goal(s). The learning process takes place through interaction with an environment (Fig. 17.1), and is therefore different from supervised learning methods which require a teacher. At each discrete time step an RL agent receives observations, i.e. an indication of the current state s. In each state s the agent can take some action a from the set of actions available in that state. An action a can cause a transition from state s to another state s', based on transition probabilities . The environment's model contains these

transition probabilities. A numerical *reward signal* r is returned to the agent to inform the RL agent about the 'goodness' of its actions or the intrinsic desirability of a state. The reward signal is also a part of the model of the environment. An RL agent searches for the optimal policy. A *policy* π maps states to actions or action probabilities. $\pi(s,a)$ denotes the probability that action a is selected in state s. An RL agent wants to maximize the expected sum of future rewards. When an infinite horizon is assumed, the discount factor γ is used to discount these future rewards. As such, less importance is given to rewards further into the future.

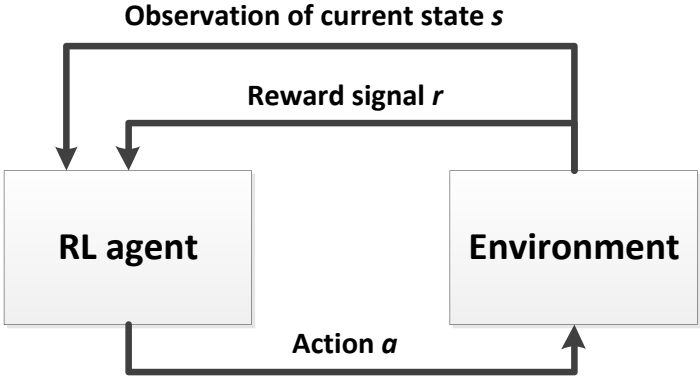

Fig. 17.1 The basic reinforcement learning model

One of the main issues in RL is balancing exploration and exploitation, i.e. whether to use the already gathered experience or to gather new experience. Another important issue is the credit assignment problem, where one have to deal with delayed rewards, and thus which action should receive credit for a given reward. The latter property is currently getting few attention in hybrid RL inspired search methods.

17.2.1 Policy Iteration Methods

When an RL method searches directly for the optimal policy in the space of policies it is called a policy iteration method. Learning Automata (LA) [24, 33] belong to this category. Other methods belonging to this type of RL algorithms are the policy gradient methods, like the REINFORCE algorithm [39]. LA are simple reinforcement learning devices that take actions in single state environments. A single learning automaton maintains an action probability distribution p, which it updates using some specific learning algorithm or reinforcement scheme. Several reinforcement schemes are available in the literature with varying convergence properties. These schemes use information from a reinforcement signal provided by the environment, and thus the LA operates with its environment in a feedback loop. Examples of

linear reinforcement schemes are linear reward-penalty, linear reward-inaction and linear reward-ε-penalty. The philosophy of these schemes is to increase the probability of selecting an action in the event of success and decrease it when the response is a failure. The general update scheme is given by:

$$\begin{aligned} p_m(t+1) &= p_m(t) + \alpha_{reward}(1-\beta(t))(1-p_m(t)) \\ &\quad - \alpha_{penalty}\beta(t)p_m(t) \\ &\quad \text{if } a_m \text{ is the action taken at time } t \end{aligned} \tag{17.1}$$

$$\begin{aligned} p_j(t+1) &= p_j(t) - \alpha_{reward}(1-\beta(t))p_j(t) \\ &\quad + \alpha_{penalty}\beta(t)[(r-1)^{-1} - p_j(t)] \\ &\quad \text{if } a_j \neq a_m \end{aligned} \tag{17.2}$$

With $p_i(t)$ the probability of selecting action i at time step t. The constants α_{reward} en $\alpha_{penalty}$ are the reward and penalty parameters. When $\alpha_{reward} = \alpha_{penalty}$, the algorithm is referred to as linear reward-penalty (L_{R-P}), when $\alpha_{penalty} = 0$, it is referred to as linear reward-inaction (L_{R-I}) and when $\alpha_{penalty}$ is small compared to α_{reward}, it is called linear reward-ε-penalty ($L_{R-\varepsilon P}$). $\beta(t)$ is the reward received by the reinforcement signal for an action taken at time step t. r is the number of actions.

17.2.2 Value Iteration Methods

Value iteration methods are more common than policy iteration methods. Value iteration methods do not directly search for optimal policy, instead they are learning evaluation functions for states or state-action pairs. Evaluation functions, as in the popular Q-learning algorithm [35, 34], can be used to evaluate the quality of a state-action pair. The Q-learning algorithm maintains an action-value function called Q-values. The Q-learning update rule is defined by

$$Q(s,a) = Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right], \tag{17.3}$$

where s is the previous state, s' the new state, a the action taken in state s, a' a possible action in state s', r the received reward, α the learning rate or step size, and γ the discount rate which indicates the importance of future rewards. In many cases the number of states or state-action pairs is too large to store, and thus the (action)-value function must be approximated. Often an artificial neural network is used to accomplish this task, but any linear or nonlinear function approximation can be used. Q-learning is known as a Temporal Difference (TD) learning method, because the update rule uses the difference between the new estimate and the old estimate of the value function. Other common TD methods are SARSA [27, 28] and TD(λ) [35].

17.2.3 Relationships between Reinforcement Learning and Metaheuristics

The link between reinforcement learning and search algorithms is not tenuous. Given a state space, an action space and a reward function, then the reinforcement learning problem can be reduced to a search in the space of policies. Thus similar issues like exploration and exploitation are faced, often called diversification and intensification in metaheuristic literature.

Unlike the subject of the present chapter where (reinforcement) learning methods are used to support metaheuristic search, there are methods which do exactly the opposite, and thus are using metaheuristic methods to improve learning. [3] show this interplay between optimization and machine learning.

17.3 Opportunities for Learning

Reinforcement learning methods can be utilized in a variety of ways to boost metaheuristic or hyper-heuristic search. Learning may help to find good settings for various parameters or components. For example, RL methods can learn properties of good *starting solutions* or an *objective function* that guides a metaheuristic towards good quality solutions. Such an approach is adopted in [4, 5], where a function is learned that is able to guide the search to good starting solutions. Another component on which learning can be applied is the *neighborhood* or *heuristic selection*. A learning method can learn which ones are the best neighborhoods or heuristics to construct or change a solution at any time during the search, such that in the end good quality solutions can be generated. Such an approach is applied in [40] and [25]. When a classical hyper-heuristic with acceptance and selection mechanism is used learning can be applied to both mechanisms. A learning method can learn to select the low-level heuristics (e.g. in [7] and [22]), or it can ascertain when to accept a move. To summarize, the possible involved components include but are not limited to:

- starting solution,
- objective function,
- neighborhoods or heuristics selection,
- acceptance of new solutions/moves.

All these parameters or components can be updated by the RL algorithm in an adaptive way.

Alternatively RL methods can also be applied directly to solve optimization problems. In other words, they are not hybridized but are themselves used as a metaheuristic. The RL method learns and directly assigns the values of the variables. Such approaches are investigated in [16], [21], [36] and [37].

A possible indication for the inclusion of RL in a search algorithm is the presence of a random component. By replacing this random component with an RL

component the algorithm develops a more intelligent decision mechanism. For example in a hyper-heuristic with a simple-random selection step, the selection can be replaced by some RL method like the one presented by [22].

Yet another opportunity for learning arises both when either the problem is intrinsically distributed or can be split into several subproblems. Examples of such include distributed scheduling and planning problems such as the decentralized resource-constrained multi-project scheduling problem (DRCMPSP) [9] and [18]. This problem considers scheduling multiple projects simultaneously, with each project having multiple jobs. A job requires local or global resources, which are available for either all jobs in the project or have to be shared among all projects respectively. Some local objectives can be optimized, for example the makespan of the individual projects, whereas global objectives, such as the average project delay or the total makespan, can be minimized. For this kind of problems multi-agent reinforcement learning methods are appropriate. When one or more global objectives need to be optimized the agents share a common goal and can thus be cooperative. The agents have to coordinate to jointly improve their decisions. Using a common reward one can simply but effectively coordinate the agents' decisions. Through sharing one reward signal the agents coordinate their decisions. This approach is applied in [36] for the DRCMPSP.

17.3.1 States and Actions

Before applying RL to a problem, the set of possible states and the set of possible actions available in each state, have to be defined. Many possible ways exist to accomplish this. First of all we can make a distinction between *search-dependent*, *problem-dependent* and *instance-dependent* state space definitions. A search-dependent state space definition uses observations of the search process itself, such as the current iteration, the number of successive non-improving iterations, or the total improvement over the initial solution. A problem-dependent setting is defined by the usage of generic problem features, like the Resource Dilation Factor for scheduling problems, as defined by [14]. An instance-dependent setting uses instance-specific features, like the number of tardy jobs in a scheduling problem, or the number of full bins in a bin-packing problem. Combinations of these three settings are also possible. When a problem-dependent or a search-dependent state space definition is used, the learned information can possibly be transfered to other instances of the same problem, or even to other problems. In many cases the properties of the solutions to the optimization problem itself cannot be used directly, due to the curse of dimensionality. Better is to use some extracted problem features. Take for example a TSP problem. If one should use the encoding of a complete tour directly as the state, then the number of states grows exponentially, i.e. $n!$ with n the number of states.

The set of possible actions in each state is defined by the set of parameters or components of the metaheuristic that one wants to learn.

17.3.2 Reward Function

Experience gathering is a prerequisite to learning, which can be achieved either online or offline. Experience in combinatorial optimization problems is scarce. Often only a single numerical value is available, indicating the quality of a complete solution. However, reward functions are very important for an RL method in order to learn some valuable information. As stated in [14], there are three requirements that a reward function should satisfy. First of all, it should give higher rewards to better solutions. Secondly, it should encourage the reinforcement learning system to find efficient search policies, i.e. search policies that involve only a few steps. Thirdly, it should be a normalized measure, in order to be transferred to new problem instances. A fourth requirement may be added; that it should be computationally efficient. When designing a reward function for a hybrid RL-metaheuristic method we do take into account these four requirements.

17.4 Literature Overview

The combination of metaheuristics or hyper-heuristics and (reinforcement) learning is relatively new. Only a limited number of papers describe a combination of the two domains. Applied problem domains include scheduling, packing and routing. Table 17.1 compares these methods by the used RL-method, metaheuristic and involved component.

One of the first papers covering the combination of learning and metaheuristic search is [40]. A reinforcement learning method is applied to learn domain-specific heuristics for the NASA space shuttle payload processing problem, which is modeled as a job shop scheduling problem. A value function is learned offline using a temporal difference algorithm TD(λ) together with a neural network. General features of the schedules (solutions) such as the percentage of the time units with a violation are used to represent a state. The possible actions are taken from a set of repair heuristics. After learning the value function on a number of small problem instances, it is used over multiple instances of the same problem. The TD algorithm is compared to an existing method for the problem, i.e. an iterative repair method with simulated annealing. The reinforcement learning based method outperforms the iterative repair method. It is noteworthy that the value functions that were learned on small problem instances also have a very good performance on larger instances. In [14] a more detailed description and application of this approach is given.

Another early contribution to the application of RL for solving combinatorial optimization problems can be found in [16]. The paper describes the Ant-Q algorithm, which combines the Ant System and the Q-Learning algorithm, and has been successfully applied to the Asymmetric Traveling Salesman Problem (ATSP). Ant System is based on the observation of ant colonies behaviour. Each ant from a colony constructs a solution for the ATSP, called a tour. The method uses a modified version of Q-values, called AQ-values. These AQ-values are updated using

a Q-learning update rule. The delayed reward, which is calculated when each ant completes a tour, is based on the best tour of the current iteration or on the best tour from all past iterations, taking into account each ant. The Ant-Q algorithm shows an interesting property. It was observed that the Ant-Q agents do not make the same tour, demonstrating the explorative character of the search method.

[21] present an algorithm that combines reinforcement learning with genetic algorithms for the Asymmetric Traveling Salesman Problem (ATSP) and Quadratic Assignment Problem (QAP). For the ATSP a Q-learning [35, 34] method is used to both express and update the desirability of choosing city a after city b. A state is a city, and an action is another city following the aforementioned city in the tour. A population of RL agents with desirability values (Q-values) is formed, with each agent holding one solution. The offspring is constructed by replicating solution parts from one parent and filling in the other parts using the desirability values (updated by a q-learning update rule) of the other parent, rather than the traditional genetic crossover operators method. As a reward a weighted combination of immediate and global rewards based on the tour lengths of the new solution and the solutions of the parents is used. The QAP is solved using a simplified update rule, that does not require a particular order as opposed to the q-learning update rule. Competitive results are shown for both addressed problems.

[4] and [5] describe the STAGE algorithm, which searches for good quality solution using two alternating phases. The first phase runs a local search method, e.g. hillclimbing or simulated annealing from a starting solution until a local optimum is reached. During this phase, the search trajectory is analyzed and used for learning an evaluation function. This is achieved by training a linear or quadratic regression method using the properties or features of the visited solutions and the objective function value of the local optimum. The authors point out that in some conditions a reinforcement learning method like TD(λ) of the temporal-difference algorithms may make better use of the training data, converge faster, and use less memory during training. The second phase performs hillclimbing on the learned evaluation function to reach a new starting solution for the first phase. This phase enables the algorithm to learn to find good starting solutions for a local search method. Empirical results are provided on seven large-scale optimization domains, e.g. bin-packing, channel routing, ... This demonstrates the ability of the STAGE algorithm to perform well on many problems.

[23] combine aspects taken from the research by [40], [4] and [5]. A reinforcement learning algorithm TD(λ) is applied to learn a value function in an offline training phase, and then uses this learned value function to solve other instances of the same problem. This method also uses features of solutions for representing a state. A linear function approximation algorithm is used. The method is applied to the dial-a-ride problem, and was compared to both the STAGE algorithm, and a 2-opt and 3-opt local search method. The method performs better than 2-opt and STAGE if the same calculation time is used. It was not as performant as 3-opt, but was a lot faster.

[25] describes a non-stationary reinforcement learning method for choosing search heuristics. At each decision point weights are used to select the search heuristics via a probabilistic selection rule (softmax) or by randomly selecting among the choices with maximal value. Based on the increase/decrease of the objective function the weights of the search heuristics are updated using simple positive/negative reinforcement rules (e.g. incrementing/decrementing the weight value). Different selection and reinforcement method combinations are tested on two types of problems - the Orc Quest problem and problems from the Logistics Domain benchmark. The author concludes that a weak positive reinforcement rule combined with a strong negative reinforcement rule works best on the tested problems.

[7] present a hyper-heuristic in which the selection of low-level heuristics makes use of basic reinforcement learning principles combined with a tabu-search mechanism. The reinforcements are performed by increasing/decreasing the rank of the low-level heuristics when the objective function value improves/worsens. The hyper-heuristic was evaluated on various instances of two distinct timetabling and rostering problems and showed to be competitive with the state-of-the-art approaches. The paper states that a key ingredient in implementing a hyper-heuristic is the learning mechanism.

An interesting study on memory length in learning hyper-heuristics is performed in [1]. Utility values or weights are used to select the low-level heuristics, similar to [25] and [7]. A dsicount factor is added to this mechanism to discount rewards later on in the search process, and thus obtaining a short term memory. The results obtained on a course timetabling problem show that a short term memory can produce better results than both no memory and infinite memory.

[15] gives an extensive overview of single and multi-agent RL approaches for distributed job-shop scheduling problems. Both value function-based and policy search-based RL methods are discussed, including policy gradient RL methods and Q-learning.

[26] present a hyper-heuristic with an RL selection mechanism and a great-deluge acceptance method for the examination timetabling problem. A set of exams must be assigned a timeslot and possibly a room while respecting a number of hard and soft constraints, such as the room capacity. An RL method based on utility values with simple update rules is used, similar to what was presented in [25]. The idea is that a heuristic is selected when it results in a lot of improving moves, and thus has a higher utility value. When a heuristic i results in an improving move the utility value u_i of that heuristic is incremented, and in case of a worsening move the utility value is lowered using three different rules, namely subtractive ($u_i = u_i - 1$), divisional ($u_i = u_i/2$) and root ($u_i = \sqrt{u_i}$). Upper and lower bounds are applied to the utility values to encourage exploration in further steps. Experiments are performed with different settings for the selection of the heuristics, the upper and lower bound, and the negative utility adaptation mechanism. The setting with a maximal selection (i.e. selecting the heuristic with a maximal utility value) and subtractive negative utility adaption mechanism performed the best. The method improves the performance of a non learning simple-random great-deluge hyper-heuristic on the examination timetabling problem.

Table 17.1 Comparison of hybrid RL-metaheuristic methods

Method	RL method	Metaheuristic	Involved component	Problem(s)
[40]	TD(λ)	Constructive Method	Heuristic selection	Job scheduling
[16]	Q-learning	Ant system	Direct	ATSP
[21]	Q-learning	GA	Direct	ATSP and QAP
[4, 5]	No RL (regression)	Local search (e.g. SA)	Obj. function, starting solution	Bin packing, channel routing, SAT,...
[23]	TD(λ)	2-opt local search	Obj. function	Dial-a-ride
[25]	Utility values	CSP solver	Heuristic selection	Orc Quest and Logistics Domain
[7]	Utility values	Hyper-heuristic	Heuristic selection	Timetabling and rostering
[1]	Utility values + discount	Hyper-heuristic	Heuristic selection	Course timetabling
[22]	LA	Hyper-heuristic	Heuristic selection	TTP
[38]	LA	GA	Direct	Project scheduling (MRCPSP)
[26]	Utility values	Hyper-heuristic	Heuristic selection	Examination timetabling
[36]	LA (+Dispersion Game)	-	Direct	Project scheduling (DRCMPSP)
[37]	LA	-	Direct	Project scheduling (MRCPSP)

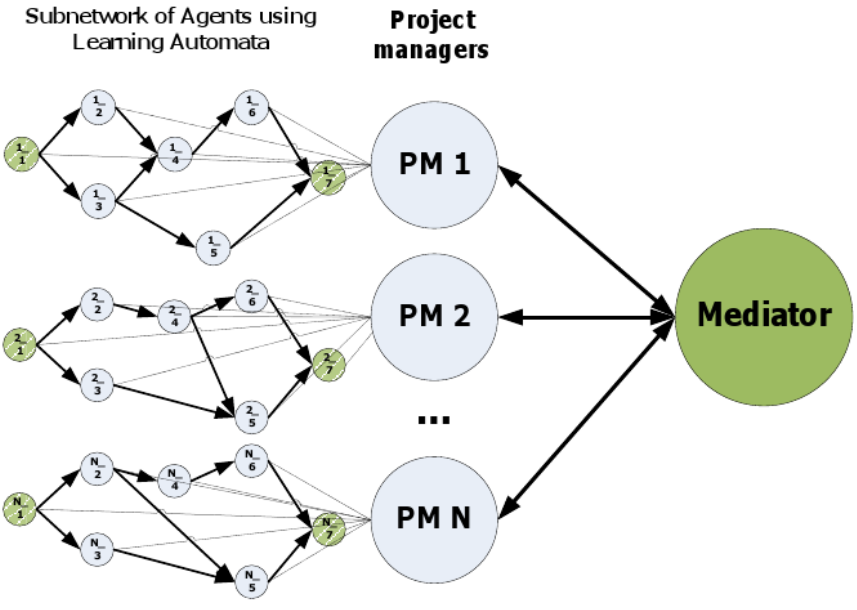

Fig. 17.2 Multi-agent system employed to solve the decentralized resource-constrained multi-project scheduling problem

Recently, learning automata have been introduced to solve combinatorial optimization problems. [22] present a heuristic selection method for hyper-heuristics, which they have applied to the traveling tournament problem. Instead of using simple reinforcement rules, a learning automaton was used for the selection. [38] describe the combination of a genetic algorithm and learning automata to solve the Multi-Mode Resource-Constrained Project Scheduling Problem (MRCPSP). The GA is applied to find good activity orders, while the LA are used to find good modes, a mode being an important decision variable of the scheduling problem. This work is extended in [37] where the GA is replaced by a network of learning automata [33]. All decision variables (i.e. activity order and modes) of the scheduling problem (MRCPSP) are now directly chosen by multiple LA. The method produces state-of-the-art results for the MRCPSP. [36] follow a very similar approach for the Decentralized Resource-Constrained Multi-Project Scheduling Problem (DRCMPSP). In the DRCMPSP multiple projects are scheduled factoring in the availability of both private and shared resources, while a global objective, i.e. the average project delay, is optimized. A network of learning automata searches for activity orders resulting in good schedules for each single project, while a dispersion game is employed to coordinate the projects. Figure 17.2 shows the multi-agent system with project managers and network of LA for solving the DRCMPSP, as applied by [36]. One motivating factor for organizing the activities in a project as learning automata is that theoretical convergence properties hold in both single and multi automata environments. One of the foundations for LA theory is that a set of decentralized learning

automata using the reward-inaction update scheme is able to control a finite Markov Chain with unknown transition probabilities and rewards. In [20], this result was extended to the framework of Markov Games. That is a straightforward extension of single-agent markov decision problems (MDP's) to distributed multi-agent decision problems. However, the convergence properties fail to hold here since the activity-on-node model does not bear the Markov property. Good results can be achieved with the network of LA in the single project scheduling scheme [37]. The methods aforementioned are added to the bottom of Table 17.1.

One might notice that most early hybrid RL-metaheuristic methods use RL algorithms as Q-learning and TD(λ) which make use of delayed rewards. Recent methods, most of them applied to hyper-heuristics, are using a more simple RL mechanism based on utility values operating in a single state environment, and thus they do not benefit the full power of RL which deals with the problem of delayed rewards and the credit assignment problem.

17.5 Best Practices

As a simple illustration of hybrid RL based systems we introduce a new learning method (LA-ILTA) using RL and show how to boost an exiting acceptance mechanism (ILTA), which was recently published [22]. We discuss possible overheads, such as extra parameters belonging to the learning components and time overhead.

17.5.1 LA-ILTA

Iteration Limited threshold acceptance (ILTA) is an acceptance mechanism for meta- and hyper-heuristics, introduced by [22]. ILTA is based on the improving or equal (IE) acceptance criterion, which only accepts non-worsening moves. Like every good acceptance criterion, ILTA tries to efficiently balance intensification and diversification. In addition to IE, ILTA accepts worsening moves under certain conditions, i.e. it accepts a move if k consecutive worsening moves are generated, and the new solution's fitness is within a certain range R of the current best solution's fitness. These two parameters k and R are fixed during the complete search. We now propose a method, called LA-ILTA, which uses RL and more specifically Learning Automata to adaptively change and learn good parameter values for ILTA. We have chosen to learn the R values, based on the place in the search process (e.g. beginning, middle or end of the search). The method thus belongs to the category of methods that use a search-dependent state space representation. We first define the state and action spaces for the RL component. We divide the search process into 10 separate periods, each one lasting 10% of the search duration. The start of each period is a state. We define the R values to be the actions possibly in each state. Thus a chosen R value is used for the next 10% of the search. The chosen discrete R values (and thus actions) are $\{1.0, 1.1, 1.2, 1.3, 1.4, 1.5\}$. In total we have 10 states and 6 actions. The

reward function, which expresses the learning goal, is the percentage improvement realized by using the chosen R value in the past search moment. In each state we use one learning automaton with LRI update scheme to select the R values. LA-ILTA is applied to the Patient Admission Scheduling (PAS) problem [12, 13]. Current best results are presented in [8]. Patients in a hospital have to be assigned to beds such that multiple hard and soft constraints regarding hospital regulations and patient preferences are met. A weighted objective function including a term for each soft constraint must be minimized. Thirteen problem instances are available [11].

We perform the following experiment. For each PAS problem instance we compare the learning LA-ILTA method to the static ILTA. For the LA-ILTA method we perform 1000 learning runs, each run from a different starting solution. Then we perform 1000 validation runs also starting from different starting solutions. For ILTA we perform only 1000 validation runs because ILTA does not include learning. We run ILTA for each static R setting $\{1.0, 1.1, 1.2, 1.3, 1.4, 1.5\}$. During these experiments all runs perform $500,000$ iterations and the k value was fixed to 100. The LA use a learning rate $\alpha_{reward} = 0.1$ and a linear reward-inaction update scheme. Table 17.2 shows the results of these experiments on the second problem instance of the PAS problem. A comparison is made between LA-ILTA and six static ILTA versions in terms of best, average and worst objective function value over 1000 validation runs. It is clear that the static ILTA with $R = 1.0$ outperforms the other static ILTA versions. However, the learning LA-ILTA method which learns a parameter setting that is dependent on the current search progress performs even better, and thus boosts the original ILTA method. All methods started from the same set of 1000 initial solutions. Similar results were obtained for the other problem instances.

Figure 17.3 shows the evolution of the objective function value over the 1000 validation runs for LA-ILTA on the second PAS problem instance. A moving average is also shown. The figure clearly shows that solutions with better objective values are reached when more learning runs are performed.

When we examine the learned policy of the LA-ILTA method, we notice that it favours higher R values (1.5) at the start of the search and lower R values (1.0) for the rest of the search progress. A transition from 1.5 to 1.0 is observed early in the search. In the beginning the learned policy allows for a lot of diversification, while later on it chooses to have more intensification. We can find a similar diversification/intensification strategy in the popular simulated annealing acceptance criterion. Figure 17.4 shows the evolution of the selected R values for the first 10% (state 1), middle 10% (state 5), and last 10% (state 10) of the search. The evolution in states 2, 3, 4, 6, 7 and 8 are ommitted from the figure for clarity. The R value for the first

Table 17.2 LA-ILTA compared to six static ILTA versions over 1000 validation runs on the second PAS problem instance

	LA-ILTA	**ILTA-1.0**	**ILTA-1.1**	**ILTA-1.2**	**ILTA-1.3**	**ILTA-1.4**	**ILTA-1.5**
Best obj.	**14898**	14960	24168	24656	24998	25220	25884
Average obj.	**16200.1**	16233.9	26796.9	27732.1	27746.4	27964.4	28610.4
Worst obj.	**17776**	18100	29272	30448	30268	30300	30842

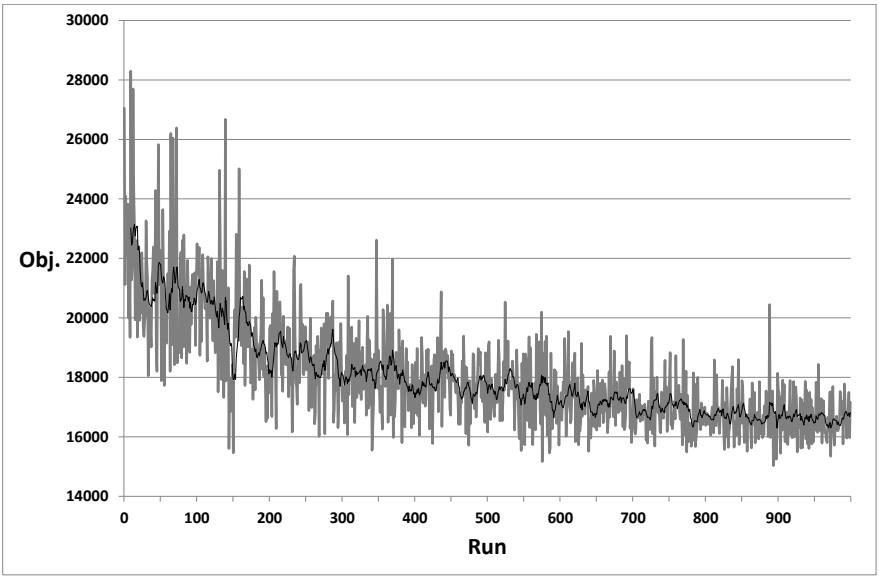

Fig. 17.3 Learning curve for LA-ILTA on the second PAS problem instance

part of the search converges rapidly to a value of 1.5, which is the highest value in the range. The middle and the last state move towards a value of 1.0, but the value for the middle state converges faster than the last state, be it slower than the first state. In general, learning in the first states goes faster than in the last states, because more information is available in the beginning of the search than at the end. This idea was also discussed in [1].

We also have applied LA-ILTA to other problems, such as the Edge Matching Puzzle (EMP) problem [10]. The problem consists of placing $n \times n$ square tiles on a board of size $n \times n$. A tile has four edges, each edge containing a pattern from a set of available patterns. All tiles must be rotated and placed on the board, such that the shared edge between neighboring tiles has a matching pattern. A special pattern (pattern 0) must only occur on the outer edges of the board. Figure 17.5 shows the results of LA-ILTA over 1000 validation runs on an Edge Matching Puzzle problem of size 10×10. Each run performing $100,000$ iterations. The LA use a learning rate $\alpha_{reward} = 0.1$ and a linear reward-inaction update scheme. Higher scores are better. The LA-ILTA method is at least as good as the best static ILTA methods. The learned policy shows similar characteristics as the learned policy on the PAS problems, i.e. high diversification in the beginning of the search process and more intensification at the end. However, unlike in the PAS experiments, the diversification does not fade away completely.

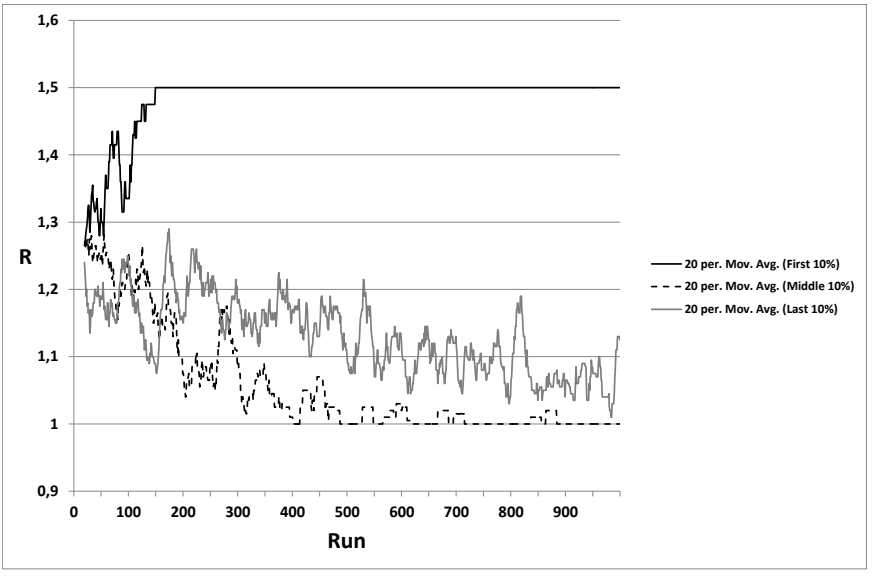

Fig. 17.4 Evolution of the selected R value setting for the first, middle and last 10% of the search on the second PAS problem instance

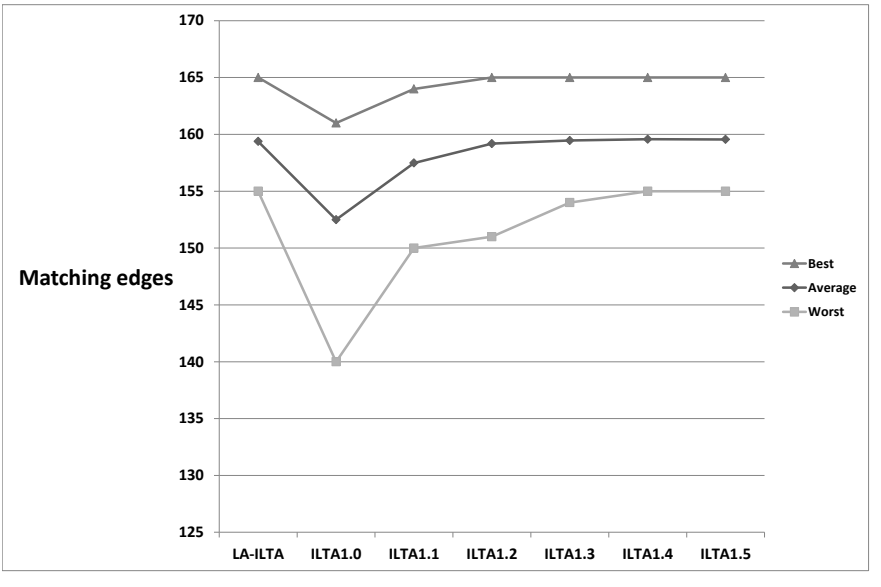

Fig. 17.5 LA-ILTA compared to six static ILTA versions over 1000 validation runs on an edge matching puzzle problem of size 10×10

17.5.2 Learning Rate

By using RL algorithms we introduce some new parameters, including for example the learning rate, the update scheme, etc. In the following experiment we study the influence of the learning rate. This parameter determines how fast the learning will proceed.

Figure 17.6 shows the learning curve (as a moving average over 20 runs) for LA-ILTA with different learning rates $\alpha_{reward} = \{0.5, 0.1, 0.05, 0.01\}$. The second PAS problem instance was used, but again similar results were observed on the other instances. The figure shows, as expected, that a higher learning rate ($\alpha_{reward} = 0.5$) leads to much faster convergence than a low learning rate ($\alpha_{reward} = 0.01$). In this example all except one learning methods converged to a similar strategy when they were given enough time to converge. The highest learning rate ($\alpha_{reward} = 0.5$) converged to a slightly different R value for the first phase. High learning rates can converge too quickly and reach a suboptimal policy. In order to select an appropriate learning rate one can count how many times each action was tried. If all actions were performed a significant number of times, then the learning rate appears to be low enough to avoid premature convergence. The influence of the learning rate on the results of a hybrid RL-metaheuristic method has to be carefully examined in the future.

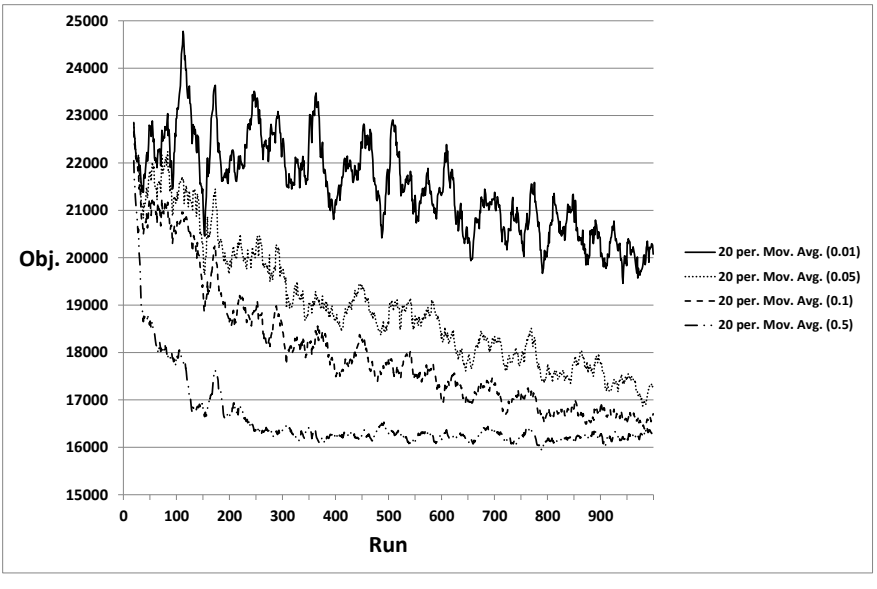

Fig. 17.6 The learning curve (moving average over 20 runs) for LA-ILTA with different learning rates $\alpha_{reward} = \{0.5, 0.1, 0.05, 0.01\}$ on the second PAS problem instance

Table 17.3 Average calculation time overhead in percentage introduced by the LA-ILTA method. Tested on 12 PAS problem instances and the eternity 2 puzzle problem.

PAS1	PAS2	PAS3	PAS4	PAS5	PAS6	PAS7	PAS8	PAS9	PAS10	PAS11	PAS12	EMP
0.41%	2.65%	1.45%	2.45%	0.38%	1.35%	4.68%	7.00%	5.40%	5.84%	9.58%	8.11%	0.31 %

17.5.3 Calculation Time Overhead

In this experiment we will investigate the calculation time overhead introduced by the reinforcement learning component. Table 17.3 shows the average calculation time overhead in % introduced by the reinforcement learning component in the LA-ILTA method on 12 PAS problem instances and an edge matching puzzle (EMP) problem instance of size 16 by 16. The LA-ILTA method uses 10 LRI learning automata, each having 6 actions. The results in the table show that the RL component never introduces more than 10% overhead to the calculation time. The overhead is hard to measure, since it is subject to various factors, such as implementation details, the metaheuristic, the RL method, the problem type and the problem size. The calculation time of the RL methods are mostly determined by the number of actions which can be selected at each decision point. Figure 17.7 shows the calculation time in milliseconds to perform 1 million selections and updates for an LRI learning automaton on a modern desktop pc with Intel Core I7-2600 3.4Ghz CPU. The calculation time grows linearly with the number of actions. For good performance, one should keep the number of actions as small as possible. The number of

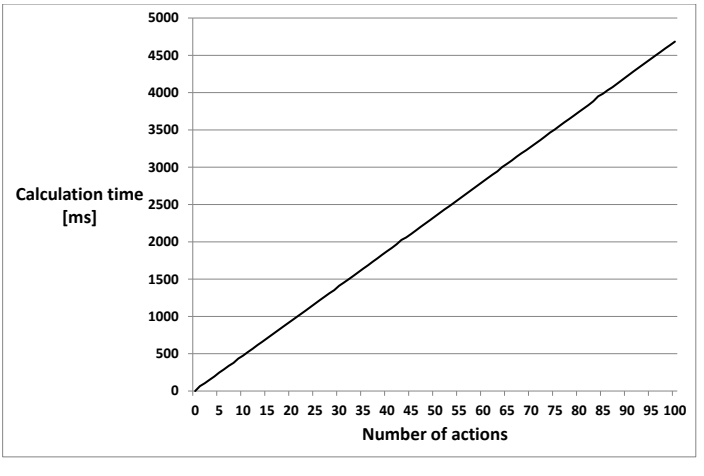

Fig. 17.7 Calculation time in ms for an LRI learning automaton to perform 1 million selections and updates, for a varying number of actions

states has no impact on the calculation time, since only one state is being updated at a time. However, the number of state transitions determines how many action selections and updates are performed, and thus affects the calculation time. The memory requirements on the other hand are determined by the product of the number of states and the number of actions.

17.6 Conclusion and Suggestions for Future Research

In the present chapter we have discussed the combination of reinforcement learning and metaheuristic search. This hybridization of two well studied research topics shows some promising methods and many opportunities for future research directions. We have discussed some main reinforcement learning components in detail, together with a literature overview on hybrid RL-metaheuristic methods. We have illustrated some examples where learning automata are able to boost the performance of metaheuristic search. An example of a new application of reinforcement learning to meta- or hyper-heuristic methods is also given, i.e. a learning acceptance method called LA-ILTA. LA-ILTA uses several learning automata to learn a search-dependent parameter value, which was used in an existent acceptance method (ILTA). This simple method was able to boost the results of the original ILTA on two tested benchmark problems, i.e. the patient admission scheduling problem and the edge matching puzzle problem. The overhead in terms of calculation time and extra parameters introduced by the RL components was studied. We have shown that simple RL methods, called learning automata, introduce little overhead.

Academics in the metaheuristic community are searching for advanced metaheuristics which enforce a particular behaviour during search. This behaviour is often determined by parameter settings which require extensive fine-tuning for each different problem. Metaheuristics equipped with (reinforcement) learning are capable of finding such a well performing behaviour themselves. Reinforcement learning methods are easy to apply. Because they make use of simple update rules, they require little extra calculation time.

Interesting directions for future research include; new RL-metaheuristic hybridizations, the usage of transfer learning methods in metaheuristic search, simultaneous learning of multiple parameters/components, multi-agent RL for decentralized problems, and RL for multi-objective optimization. To give a better boost to metaheuristic search, the full power of RL should be used, i.e. incorporate a mechanism of delayed reward or go beyond single state learning.

Acknowledgements. We thank Erik Van Achter for his help on improving the quality of this text. We also thank Wim Vancroonenburg for providing the basic setup for the PAS problem experiments.

References

1. Bai, R., Burke, E.K., Gendreau, M., Kendall, G., Mccollum, B.: Memory length in hyper-heuristics: An empirical study. In: Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Scheduling, CI-Sched 2007 (2007)
2. Battiti, R., Brunato, M., Mascia, F.: Reactive Search and Intelligent Optimization. Operations research/Computer Science Interfaces, vol. 45. Springer (2008)
3. Bennett, K.P., Parrado-Hernández, E.: The interplay of optimization and machine learning research. J. Mach. Learn. Res. 7, 1265–1281 (2006)
4. Boyan, J.: Learning Evaluation Functions for Global Optimization. PhD thesis, Carnegie-Mellon University (1998)
5. Boyan, J., Moore, A.W., Kaelbling, P.: Learning evaluation functions to improve optimization by local search. Journal of Machine Learning Research 1, 2000 (2000)
6. Burke, E., Hart, E., Kendall, G., Newall, J., Ross, P., Schulenburg, S.: Hyper-heuristics: An emerging direction in modern search technology. In: Handbook of Metaheuristics, pp. 457–474. Kluwer Academic Publishers (2003)
7. Burke, E.K., Kendall, G., Soubeiga, E.: A tabu-search hyperheuristic for timetabling and rostering. Journal of Heuristics 9, 451–470 (2003)
8. Ceschia, S., Schaerf, A.: Local search and lower bounds for the patient admission scheduling problem. Computers & Operartions Research 38, 1452–1463 (2011)
9. Confessore, G., Giordani, S., Rismondo, S.: A market-based multi-agent system model for decentralized multi-project scheduling. Annals of Operational Research 150, 115–135 (2007)
10. Demaine, E.D., Demaine, M.L.: Jigsaw puzzles, edge matching, and polyomino packing: Connections and complexity. Graphs and Combinatorics 23, 195–208 (2007); Special issue on Computational Geometry and Graph Theory: The Akiyama-Chvatal Festschrift
11. Demeester. P.: Patient admission scheduling website (2009), `http://allserv.kahosl.be/~peter/pas/` (last visit August 15, 2011)
12. Demeester, P., De Causmaecker, P., Vanden Berghe, G.: Applying a local search algorithm to automatically assign patients to beds. In: Proceedings of the 22nd Conference on Quantitive Decision Making (Orbel 22), pp. 35–36 (2008)
13. Demeester, P., Souffriau, W., De Causmaecker, P., Vanden Berghe, G.: A hybrid tabu search algorithm for automatically assigning patients to beds. Artif. Intell. Med. 48, 61–70 (2010)
14. Dietterich, T.G., Zhang, W.: Solving combinatorial optimization tasks by reinforcement learning: A general methodology applied to resource-constrained scheduling. Journal of Artificial Intelligence Research (2000)
15. Gabel, T.: Multi-agent Reinforcement Learning Approaches for Distributed Job-Shop Scheduling Problems. PhD thesis, Universität Osnabrück, Deutschland (2009)
16. Gambardella, L.M., Dorigo, M.: Ant-q: A réinforcement learning approach to the traveling salesman problem, pp. 252–260. Morgan Kaufmann (1995)
17. Glover, F., Kochenberger, G.A.: Handbook of metaheuristics. Springer (2003)
18. Homberger, J.: A (μ, λ)-coordination mechanism for agent-based multi-project scheduling. OR Spectrum (2009), doi:10.1007/s00291-009-0178-3
19. Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement learning: A survey. Journal of Artificial Intelligence Research 4, 237–285 (1996)
20. Littman, M.L.: Markov games as a framework for multi-agent reinforcement learning. In: Proceedings of the Eleventh International Conference on Machine Learning, pp. 157–163. Morgan Kaufmann (1994)
21. Miagkikh, V.V., Punch III, W.F.: An approach to solving combinatorial optimization problems using a population of reinforcement learning agents (1999)
22. Misir, M., Wauters, T., Verbeeck, K., Vanden Berghe, G.: A new learning hyper-heuristic for the traveling tournament problem. In: Proceedings of Metaheuristic International Conference (2009)

23. Moll, R., Barto, A.G., Perkins, T.J., Sutton, R.S.: Learning instance-independent value functions to enhance local search. In: Advances in Neural Information Processing Systems, pp. 1017–1023. MIT Press (1998)
24. Narendra, K., Thathachar, M.: Learning Automata: An Introduction. Prentice-Hall International, Inc. (1989)
25. Nareyek, A.: Choosing search heuristics by non-stationary reinforcement learning. In: Metaheuristics: Computer Decision-Making, pp. 523–544. Kluwer Academic Publishers (2001)
26. Özcan, E., Misir, M., Ochoa, G., Burke, E.K.: A reinforcement learning - great-deluge hyper-heuristic for examination timetabling. Int. J. of Applied Metaheuristic Computing, 39–59 (2010)
27. Rummery, G.A., Niranjan, M.: On-line q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166, Engineering Department, Cambridge University (1994)
28. Richard, S., Sutton, R.S.: Generalization in reinforcement learning: Successful examples using sparse coarse coding. In: Touretzky, D.S., Mozer, M.C., Hasselmo, M.E. (eds.) Advances in Neural Information Processing Systems: Proceedings of the 1995 Conference, pp. 1038–1044. MIT Press, Cambridge (1996)
29. Sutton, R.S., Barto, A.G.: Reinforcement learning: an introduction. MIT Press (1998)
30. Talbi, E.-G.: Metaheuristics: From Design to Implementation. John Wiley and Sons (2009)
31. Taylor, M.E., Stone, P.: Transfer learning for reinforcement learning domains: A survey. J. Mach. Learn. Res. 10, 1633–1685 (2009)
32. Taylor, M.E., Stone, P., Liu, Y.: Transfer learning via inter-task mappings for temporal difference learning. Journal of Machine Learning Research 8(1), 2125–2167 (2007)
33. Thathachar, M.A.L., Sastry, P.S.: Networks of Learning Automata: Techniques for Online Stochastic Optimization. Kluwer Academic Publishers (2004)
34. Watkins, C.J.C.H., Dayan, P.: Q-learning. Machine Learning 8, 279–292 (1992)
35. Watkins, C.J.C.H.: Learning from Delayed Rewards. PhD thesis, Cambridge University (1989)
36. Wauters, T., Verbeeck, K., De Causmaecker, P., Vanden Berghe, G.: A game theoretic approach to decentralized multi-project scheduling (extended abstract). In: Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems, AAMAS 2010, vol. R24 (2010)
37. Wauters, T., Verbeeck, K., Vanden Berghe, G., De Causmaecker, P.: Learning agents for the multi-mode project scheduling problem. Journal of the Operational Research Society 62(2), 281–290 (2011)
38. Wauters, T., Verstichel, J., Verbeeck, K., Vanden Berghe, G.: A learning metaheuristic for the multi mode resource constrained project scheduling problem. In: Proceedings of the Third Learning and Intelligent OptimizatioN Conference, LION3 (2009)
39. Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning, 229–256 (1992)
40. Zhang, W., Dietterich, T.: A reinforcement learning approach to job-shop scheduling. In: Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, pp. 1114–1120. Morgan Kaufmann (1995)